



UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/226,939	01/08/1999	JOHN K. VINCENT	346872000500	8916

24024 7590 07/19/2005

CALFEE HALTER & GRISWOLD, LLP
800 SUPERIOR AVENUE
SUITE 1400
CLEVELAND, OH 44114

EXAMINER

LY, ANH

ART UNIT PAPER NUMBER

2162

DATE MAILED: 07/19/2005

Please find below and/or attached an Office communication concerning this application or proceeding.

Office Action Summary

Application No.

09/226,939

Applicant(s)

VINCENT ET AL.

Examiner

Anh Ly

Art Unit

2162

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If the period for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

- 1) ☒ Responsive to communication(s) filed on 25 April 2005.
- 2a) ☐ This action is FINAL. 2b) ☒ This action is non-final.
- 3) ☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

- 4) ☒ Claim(s) 9-29 and 31-39 is/are pending in the application.
- 4a) Of the above claim(s) 1-8 and 30 is/are withdrawn from consideration.
- 5) ☐ Claim(s) _____ is/are allowed.
- 6) ☐ Claim(s) 9-29 and 31-39 is/are rejected.
- 7) ☐ Claim(s) _____ is/are objected to.
- 8) ☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

- 9) ☐ The specification is objected to by the Examiner.
- 10) ☒ The drawing(s) filed on 08 January 1999 is/are: a) ☒ accepted or b) ☐ objected to by the Examiner.
Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).
Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).
- 11) ☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

- 12) ☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).
- a) ☐ All b) ☐ Some * c) ☐ None of:
- ☐ Certified copies of the priority documents have been received.
 - ☐ Certified copies of the priority documents have been received in Application No. _____.
 - ☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

- 1) ☒ Notice of References Cited (PTO-892)
- 2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
- 3) ☐ Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08)
Paper No(s)/Mail Date _____
- 4) ☐ Interview Summary (PTO-413)
Paper No(s)/Mail Date. _____
- 5) ☐ Notice of Informal Patent Application (PTO-152)
- 6) ☐ Other: _____

DETAILED ACTION

1. This Office Action is response to Applicants' Amendment After Final filed on 04/25/2005.
2. Claims 1-8 and 30 were cancelled (dated 04/13/2004).
3. Claims 31-39 are added (dated 04/25/2005).
4. Claims 9-29 and 31-39 are pending in this application.

Claim Rejections - 35 USC § 103

5. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negated by the manner in which the invention was made.

This application currently names joint inventors. In considering patentability of the claims under 35 U.S.C. 103(a), the examiner presumes that the subject matter of the various claims was commonly owned at the time any inventions covered therein were made absent any evidence to the contrary. Applicant is advised of the obligation under 37 CFR 1.56 to point out the inventor and invention dates of each claim that was not commonly owned at the time a later invention was made in order for the examiner to

Art Unit: 2162

consider the applicability of 35 U.S.C. 103(c) and potential 35 U.S.C. 102(e), (f) or (g) prior art under 35 U.S.C. 103(a).

6. Claims 9-16 are rejected under 35 U.S.C. 103(a) as being unpatentable over US Patent No. 5,325,531 issued to McKeeman et al. (hereinafter McKeeman) in view of US Patent No. 5,826,077 issued to Blakeley et al. (hereinafter Blakeley).

With respect to claim 9, McKeeman teaches direct dependencies of a code object and then each dependency found and all basic dependencies are generated into a dependency tree (analysis the source text or code with all ox text must be recompiled or by examining a developer-prepared dependency and dependency analysis automatically generates dependency information from the application source module: abstract, col. 37, lines 20-47 and col. 38, lines 1-3).

McKeeman teaches automatically dependency graphs or dependency trees as shown in fig. 6B to identify the dependencies between symbols within the application. McKeeman does not clearly teach querying a data catalog and doing the query recursively.

However, Blakeley teaches query object-oriented, data dictionary containing all class information in object-oriented database queries and query recursively defined in terms of objects in the hierarchical or dependency graphs to represent complex objects (col. 6, lines 44-67, col. 7, lines 22-62 and also see col. 5, lines 10-18).

Therefore, it would have been obvious to a person of ordinary skill in the art at the time the invention was made to combine the teachings of McKeeman with the teachings of Blakeley, wherein the dependency graph or dependency tree is generated

Art Unit: 2162

automatically based on the source code line in the text file in the system provided therein (McKeeman's fig. 6, col. 13, lines 25-52), would incorporate the use of querying data dictionary in object-oriented code objects and query recursively in the dependency tree, in the same conventional manner as described by Blakeley (col. 6, lines 44-67 and col. 6, lines 22-62). The motivation being to generate a dependency tree based on code files of objects in the object-oriented database and query recursively the dependency tree.

With respect to claim 10, McKeeman teaches debugging tools as debugger (see col. 5, lines 18-67 and col. 6, lines 1-12).

With respect to claim 11, McKeeman teaches coverage tool (col. 2, lines 32-44; col. 6, lines 25-45; col. 5, lines 18-67 and col. 6, lines 1-12).

With respect to claim 12, McKeeman teaches profiling tool (col. 2, lines 32-44; col. 6, lines 25-45; col. 5, lines 18-67 and col. 6, lines 1-12).

With respect to claim 13, McKeeman teaches and testing tool (col. 2, lines 32-44; col. 6, lines 25-45; col. 5, lines 18-67 and col. 6, lines 1-12).

With respect to claim 14, McKeeman teaches invalid entries for a database (col. 2, lines 32-44; col. 6, lines 25-45; col. 5, lines 18-67 and col. 6, lines 1-12).

With respect to claim 15, McKeeman teaches, dependencies among database code object (col. 2, lines 32-44; col. 6, lines 25-45; col. 5, lines 18-67 and col. 6, lines 1-12;).

With respect to claim 16, McKeeman teaches a dependency graph presentation tools (tools for software development: col. 6, lines 1-12).

7. Claim 17-21, 22-26, 27-29, 31-38 and 39 are rejected under 35 U.S.C. 103(a) as being unpatentable over US Patent No. 5,325,531 issued to McKeeman et al. (hereinafter McKeeman) in view of US Patent No. 5,8,26,077 issued to Blakeley et al. (hereinafter Blakeley) and further in view of US Patent No. 5,926,819 issued to Doo et al. (hereinafter Doo).

With respect to claim 17, McKeeman teaches dependencies of a code object and then each dependency found and all basic dependencies are generated into a dependency tree (analysis the source text or code with all ox text must be recompiled or by examining a developer-prepared dependency and dependency analysis automatically generates dependency information from the application source module: abstract, col. 37, lines 20-47 and col. 38, lines 1-3).

McKeeman teaches automatically dependency graphs or dependency trees as shown in fig. 6B to identify the dependencies between symbols within the application. McKeeman does not clearly teach using a recursive algorithm for querying a database catalog, applying the recursive algorithm on each of the dependencies into the dependency graph.

However, Blakeley teaches query object-oriented, data dictionary containing all class information in object-oriented database queries and query recursively defined in terms of objects in the hierarchical or dependency graphs to represent complex objects (col. 6, lines 44-67, col. 7, lines 22-62 and also see col. 5, lines 10-18). Also Blakeley teaches parsing module for the code object in the dependency graph.

Therefore, based on McKeeman in view of Blakeley, it would have been obvious to a person of ordinary skill in the art at the time the invention was made to utilize the teachings of Blakeley to the system of McKeeman for query recursively the dependency code object for the dependency graph. McKeeman and Blakeley do not teach using a parser on each of code objects in the dependency to identify DML statement that "fire" triggers so as to identify dependencies on triggers.

However, Doo teaches DML statement being applied to fire the triggers (col. 5, lines 22-42, also see figs. 2-4).

Therefore, based on McKeeman in view of Blakeley, and further in view of Doo, it would have been obvious to a person of ordinary skill in the art at the time the invention was made to combine the teachings of Doo to the system of McKeeman for identifying DML statements that triggers so as to identify dependencies on triggers. The motivation being to generate a dependency tree based on code files of objects in the object-oriented database and query recursively the dependency tree.

With respect to claim 18, McKeeman teaches coverage tool (col. 2, lines 32-44; col. 6, lines 25-45; col. 5, lines 18-67 and col. 6, lines 1-12).

With respect to claim 19, McKeeman teaches profiling tool (col. 2, lines 32-44; col. 6, lines 25-45; col. 5, lines 18-67 and col. 6, lines 1-12).

With respect to claim 20, McKeeman teaches and testing tool (col. 2, lines 32-44; col. 6, lines 25-45; col. 5, lines 18-67 and col. 6, lines 1-12).

With respect to claim 21, McKeeman teaches invalid entries for a database (col. 2, lines 32-44; col. 6, lines 25-45; col. 5, lines 18-67 and col. 6, lines 1-12).

With respect to claim 22, McKeeman teaches outputs a direct dependency graph of a database code object, the "direct dependency graph" containing dependencies of object-oriented code objects in the database (source code and code table: col. 5, lines 18-55 and col. 6, lines 1-12, analysis the source text or code with all ox text must be recompiled or by examining a developer-prepared dependency and dependency analysis automatically generates dependency information from the application source module: abstract, col. 37, lines 20-47 and col. 38, lines 1-3).

McKeeman teaches automatically dependency graphs or dependency trees as shown in fig. 6B to identify the dependencies between symbols within the application. McKeeman does not clearly teach applying a recursive algorithm that queries a database, applying the recursive algorithm on each of the object oriented code objects in the dependency graph.

However, Blakeley teaches query object-oriented, data dictionary containing all class information in object-oriented database queries and query recursively defined in terms of objects in the hierarchical or dependency graphs to represent complex objects (col. 3, lines 52-67 col. 4, lines 20-25 and lines 42-55; col. 6, lines 44-67, col. 7, lines 22-62 and also see col. 5, lines 10-18). Also Blakeley teaches parsing module for the code object in the dependency graph.

Therefore, based on McKeeman in view of Blakeley, it would have been obvious to a person of ordinary skill in the art at the time the invention was made to utilize the teachings of Blakeley to the system of McKeeman for query recursively the dependency

Art Unit: 2162

code object for the dependency graph. McKeeman and Blakeley do not teach involve dependencies on triggers.

However, Doo teaches DML statement being applied to fire the triggers (col. 5, lines 22-42, also see figs. 2-4).

Therefore, based on McKeeman in view of Blakeley, and further in view of Doo, it would have been obvious to a person of ordinary skill in the art at the time the invention was made to combine the teachings of Doo to the system of McKeeman for identifying DML statements that triggers so as to identify dependencies on triggers. The motivation being to generate a dependency tree based on code files of objects in the object-oriented database and query recursively the dependency tree.

With respect to claim 23, McKeeman teaches coverage tool (col. 2, lines 32-44; col. 6, lines 25-45; col. 5, lines 18-67 and col. 6, lines 1-12).

With respect to claim 24, McKeeman teaches profiling tool (col. 2, lines 32-44; col. 6, lines 25-45; col. 5, lines 18-67 and col. 6, lines 1-12).

With respect to claim 25, McKeeman teaches and testing tool (col. 2, lines 32-44; col. 6, lines 25-45; col. 5, lines 18-67 and col. 6, lines 1-12).

With respect to claim 26, McKeeman teaches invalid entries for a database (col. 2, lines 32-44; col. 6, lines 25-45; col. 5, lines 18-67 and col. 6, lines 1-12).

With respect to claim 27, McKeeman teaches generating a dependency graph, the dependency graph being a data structure and having entries to contain representations of depending code objects, specifications of packages (source code and code table: col. 5, lines 18-55 and col. 6, lines 1-12, analysis the source text or

Art Unit: 2162

code with all ox text must be recompiled or by examining a developer-prepared dependency and dependency analysis automatically generates dependency information from the application source module: abstract, col. 37, lines 20-47 and col. 38, lines 1-3).

McKeeman teaches automatically dependency graphs or dependency trees as shown in fig. 6B to identify the dependencies between symbols within the application. McKeeman does not clearly teach a code mechanism for querying a database for dependency information.

However, Blakeley teaches query object-oriented, data dictionary containing all class information in object-oriented database queries and query recursively defined in terms of objects in the hierarchical or dependency graphs to represent complex objects (col. 3, lines 52-67 col. 4, lines 20-25 and lines 42-55; col. 6, lines 44-67, col. 7, lines 22-62 and also see col. 5, lines 10-18). Also Blakeley teaches relational database for the code object in the dependency graph.

Therefore, based on McKeeman in view of Blakeley, it would have been obvious to a person of ordinary skill in the art at the time the invention was made to utilize the teachings of Blakeley to the system of McKeeman for query recursively the dependency code object for the dependency graph. McKeeman and Blakeley do not teach a digital computer, database server and implements of types and triggers and dependencies of triggers.

However, Doo teaches computer system, database system and database server (fig.1, col. 4, lines 1-20 and col. 5, lines 60-67); DML statement being applied to fire the triggers (col. 5, lines 22-42, also see figs, 2-4).

Therefore, based on McKeeman in view of Blakeley, and further in view of Doo, it would have been obvious to a person of ordinary skill in the art at the time the invention was made to combine the teachings of Doo to the system of McKeeman for identifying DML statements that triggers so as to identify dependencies on triggers. The motivation being to generate a dependency tree based on code files of objects in the object-oriented database and query recursively the dependency tree.

With respect to claim 28, McKeeman teaches generating a dependency graph, the dependency graph being a data structure and having entries to contain representations of depending code objects, specifications of packages (source code and code table: col. 5, lines 18-55 and col. 6, lines 1-12, analysis the source text or code with all ox text must be recompiled or by examining a developer-prepared dependency and dependency analysis automatically generates dependency information from the application source module: abstract, col. 37, lines 20-47 and col. 38, lines 1-3).

McKeeman teaches automatically dependency graphs or dependency trees as shown in fig. 6B to identify the dependencies between symbols within the application. McKeeman does not clearly teach a code mechanism for querying a database for dependency information.

However, Blakeley teaches query object-oriented, data dictionary containing all class information in object-oriented database queries and query recursively defined in terms of objects in the hierarchical or dependency graphs to represent complex objects (col. 3, lines 52-67 col. 4, lines 20-25 and lines 42-55; col. 6, lines 44-67, col. 7, lines

22-62 and also see col. 5, lines 10-18). Also Blakeley teaches relational database for the code object in the dependency graph.

Therefore, based on McKeeman in view of Blakeley, it would have been obvious to a person of ordinary skill in the art at the time the invention was made to utilize the teachings of Blakeley to the system of McKeeman for query recursively the dependency code object for the dependency graph. McKeeman and Blakeley do not teach triggers, and dependencies of triggers.

However, Doo teaches DML statement being applied to fire the triggers (col. 5, lines 22-42, also see figs. 2-4).

Therefore, based on McKeeman in view of Blakeley, and further in view of Doo, it would have been obvious to a person of ordinary skill in the art at the time the invention was made to combine the teachings of Doo to the system of McKeeman for identifying DML statements that triggers so as to identify dependencies on triggers. The motivation being to generate a dependency tree based on code files of objects in the object-oriented database and query recursively the dependency tree.

With respect to claim 29, McKeeman teaches generating a dependency graph, the dependency graph being a data structure and having entries to contain representations of depending code objects, specifications of packages (source code and code table: col. 5, lines 18-55 and col. 6, lines 1-12, analysis the source text or code with all ox text must be recompiled or by examining a developer-prepared dependency and dependency analysis automatically generates dependency information

Art Unit: 2162

from the application source module: abstract, col. 37, lines 20-47 and col. 38, lines 1-3) and debugging software (col. 2, lines 61-67 and col. 3, lines 1-15).

McKeeman teaches automatically dependency graphs or dependency trees as shown in fig. 6B to identify the dependencies between symbols within the application. McKeeman does not clearly teach a recursive code mechanism for querying a database for dependency information, a code mechanism for using the dependency graph to debug the target.

However, Blakeley teaches query object-oriented, data dictionary containing all class information in object-oriented database queries and query recursively defined in terms of objects in the hierarchical or dependency graphs to represent complex objects (col. 3, lines 52-67 col. 4, lines 20-25 and lines 42-55; col. 6, lines 44-67, col. 7, lines 22-62 and also see col. 5, lines 10-18). Also Blakeley teaches relational database for the code object in the dependency graph.

Therefore, based on McKeeman in view of Blakeley, it would have been obvious to a person of ordinary skill in the art at the time the invention was made to utilize the teachings of Blakeley to the system of McKeeman for query recursively the dependency code object for the dependency graph. McKeeman and Blakeley do not teach implements of type types and triggers and dependencies of triggers.

However, Doo teaches DML statement being applied to fire the triggers (col. 5, lines 22-42, also see figs. 2-4).

Therefore, based on McKeeman in view of Blakeley, and further in view of Doo, it would have been obvious to a person of ordinary skill in the art at the time the invention

Art Unit: 2162

was made to combine the teachings of Doo to the system of McKeeman for identifying DML statements that triggers so as to identify dependencies on triggers. The motivation being to generate a dependency tree based on code files of objects in the object-oriented database and query recursively the dependency tree.

With respect to claim 31, McKeeman teaches outputs a direct dependency graph of a database code object, the "direct dependency graph" containing dependencies of object-oriented code objects in the database (source code and code table: col. 5, lines 18-55 and col. 6, lines 1-12, analysis the source text or code with all ox text must be recompiled or by examining a developer-prepared dependency and dependency analysis automatically generates dependency information from the application source module: abstract, col. 37, lines 20-47 and col. 38, lines 1-3).

McKeeman teaches automatically dependency graphs or dependency trees as shown in fig. 6B to identify the dependencies between symbols within the application. McKeeman does not clearly teach applying a recursive algorithm that queries a database, applying the recursive algorithm on each of the object oriented code objects in the dependency graph.

However, Blakeley teaches query object-oriented, data dictionary containing all class information in object-oriented database queries and query recursively defined in terms of objects in the hierarchical or dependency graphs to represent complex objects (col. 3, lines 52-67 col. 4, lines 20-25 and lines 42-55; col. 6, lines 44-67, col. 7, lines 22-62 and also see col. 5, lines 10-18). Also Blakeley teaches parsing module for the code object in the dependency graph.

Therefore, based on McKeeman in view of Blakeley, it would have been obvious to a person of ordinary skill in the art at the time the invention was made to utilize the teachings of Blakeley to the system of McKeeman for query recursively the dependency code object for the dependency graph. McKeeman and Blakeley do not teach involve dependencies on triggers.

However, Doo teaches DML statement being applied to fire the triggers (col. 5, lines 22-42; also see figs. 2-4).

Therefore, based on McKeeman in view of Blakeley, and further in view of Doo, it would have been obvious to a person of ordinary skill in the art at the time the invention was made to combine the teachings of Doo to the system of McKeeman for identifying DML statements that triggers so as to identify dependencies on triggers. The motivation being to generate a dependency tree based on code files of objects in the object-oriented database and query recursively the dependency tree.

With respect to claim 32, McKeeman teaches using the generated dependency graph to compile code objects in debug mode as part of a database code object debugging tool (fig. 1 and col. 6, lines 3-22).

With respect to claim 33, McKeeman teaches using the generated dependency graph to identify calling paths in a database code coverage tool (col. 2, lines 32-44; col. 6, lines 25-45; col. 5, lines 18-67 and col. 6, lines 1-12).

With respect to claim 34, McKeeman teaches using the generated dependency graph to identify calling paths in a database code object profiling tool (col. 2, lines 32-44; col. 6, lines 25-45; col. 5, lines 18-67 and col. 6, lines 1-12).

With respect to claim 35, McKeeman teaches using the generated dependency graph in a database code object testing tool (col. 2, lines 32-44; col. 6, lines 25-45; col. 5, lines 18-67 and col. 6, lines 1-12).

With respect to claim 36, McKeeman teaches using the generated dependency graph to identify dependent objects that are INVALID in the database (col. 2, lines 32-44; col. 6, lines 25-45; col. 5, lines 18-67 and col. 6, lines 1-12).

With respect to claim 37, McKeeman teaches using the generated dependency graph to identify cyclic dependencies among database code objects (col. 2, lines 32-44; col. 6, lines 25-45; col. 5, lines 18-67 and col. 6, lines 1-12).

With respect to claim 38, McKeeman teaches using the generated dependency graph in a dependency graph presentation tool (col. 2, lines 32-44; col. 6, lines 25-45; col. 5, lines 18-67 and col. 6, lines 1-12).

With respect to claim 39, McKeeman teaches outputs a direct dependency graph of a database code object, the "direct dependency graph" containing dependencies of object-oriented code objects in the database (source code and code table: col. 5, lines 18-55 and col. 6, lines 1-12, analysis the source text or code with all ox text must be recompiled or by examining a developer-prepared dependency and dependency analysis automatically generates dependency information from the application source module: abstract, col. 37, lines 20-47 and col. 38, lines 1-3).

McKeeman teaches automatically dependency graphs or dependency trees as shown in fig. 6B to identify the dependencies between symbols within the application. McKeeman does not clearly teach applying a recursive algorithm that queries a

database, applying the recursive algorithm on each of the object oriented code objects in the dependency graph.

However, Blakeley teaches query object-oriented, data dictionary containing all class information in object-oriented database queries and query recursively defined in terms of objects in the hierarchical or dependency graphs to represent complex objects (col. 3, lines 52-67 col. 4, lines 20-25 and lines 42-55; col. 6, lines 44-67, col. 7, lines 22-62 and also see col. 5, lines 10-18). Also Blakeley teaches parsing module for the code object in the dependency graph.

Therefore, based on McKeeman in view of Blakeley, it would have been obvious to a person of ordinary skill in the art at the time the invention was made to utilize the teachings of Blakeley to the system of McKeeman for query recursively the dependency code object for the dependency graph. McKeeman and Blakeley do not teach involve dependencies on triggers.

However, Doo teaches DML statement being applied to fire the triggers (col. 5, lines 22-42, also see figs. 2-4).

Therefore, based on McKeeman in view of Blakeley, and further in view of Doo, it would have been obvious to a person of ordinary skill in the art at the time the invention was made to combine the teachings of Doo to the system of McKeeman for identifying DML statements that triggers so as to identify dependencies on triggers. The motivation being to generate a dependency tree based on code files of objects in the object-oriented database and query recursively the dependency tree.


Art Unit: 2162


Contact Information

8. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Anh Ly whose telephone number is (571) 272-4039 or via E-Mail: ANH.LY@USPTO.GOV or fax to **(571) 273-4039**. The examiner can normally be reached on TUESDAY – THURSDAY from 8:30 AM – 3:30 PM.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, John Breene, can be reached on (571) 272-4107 or **Primary Examiner Jean Corrielus (571) 272-4032**.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see <http://pair-direct.uspto.gov>. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). Any response to this action should be mailed to: Commissioner of Patents and Trademarks, Washington, D.C. 20231, or faxed to: Central Fax Center **(571) 273-8300**

ANH LY 
JUL. 14th, 2005


JEAN M. CORRIELUS
PRIMARY EXAMINER